* VZ-LINK *
NEWLETER

<< AUCKLAND VZ 300/200 USERS CLUB >>

ISSUE # 18 MARCH 1988

Well,here we are again with another information packed issue of LINK.
It's hard to believe that it's March already,Is'nt it.?
This issue see's the first part of a superb item written by Bob kitch,whom some of you will of heard of before.
Bobs "Introduction to programming" item seem's a fitting way to follow on from Robert Quinns series on "Understanding Your VZ."

I hope everyone is working on something to go into the Newsletter.Unfortunately there is very little local input. Every little bit help's,so have a look around and see what you can come up with.
This issue is ready to post out,just before the Meeting/workshop on the 27Th,so a full report on how it went will appear in the April issue.
To those that did attend,thank you for attending and making the whole thing worth while.
Between these meetings,I would be willing to have Informal sessions at my place on a saturday or sunday,for those users that need information or are having problem's.!
I am not a VZ expert,but sometimes two heads are better than one and I have a vast library of VZ items that may hold the answer.If you need help,Just phone to check that it's ok to come over.
In past issues,I have been very hard on Dick Smith's for the lack of support they offer us users.
Well on Friday 19Th Feb I paid a visit to their NZ Head office. There I met Stephen Macartney who is now responsible for NZ Computer sales and support.After speaking to him,I realized that the lack of support starts at the Australian end of the Dick Smith Setup.
Stephen has not only arranged for the club to borrow a monitor for use at the meeting.But is also arranging for club notices to go to all stores.
He is also trying to get more information from Australia for us.Thank you Stephen for your efforts on our behalf.

If you have any Software that you feel is worth trying to sell.Send it to me and I will market it for you.
This is a service setup to increase support,not make a profit.As long as I recover the cost of disk's & postage I am quite happy to do this.I will hold a master disk and make copies for resale.You set the price and all money from sales will be sent to you minus 10% handling charges.

SOFTWARE REVIEW

GALACTIC EMPIRES

SOFTWARE WRITTEN AND MARKETED BY SCOTT LE BRUN
AND REVIEWED BY PETER J HILL (AUCKLAND N.Z)

This program includes a very comprehensive and easy to understand scenario.The Instructions are also easy to follow. So read them and play--You will be glad you did!.

You-The power hungry emperor of a small star system ,must build up your empire and dispatch your star ship's to explore other star systems.
The object of all this exploring is to plunder and conquer these star systems and thereby increase your power and wealth to continue feeding your lust for supremacy.

You can have upto 4 players in this game or you can play the master gamer (YOUR VZ).You can also nominate how many turns you wish to play,thus avoiding a never ending game and no result.
You can even rename your home star,If you wish(note new name must start with F)

Once you have fed the VZ this data,you will be informed that the computer is working thing's out.This is a really nice touch and at least you don't start thinking the games hung up when faced with a blank screen.
By pressing the O key you will be given a commandlist/menu which will enable you to gather information,such as battle results,star ship location,star map etc.

This game is a must for the adventurous gamer and is well worth the cost,which at the time of writing this review is only A$12-50.

Galactic empires is a super space simulation where you must pit your skill and cunning to beat your opponent, who is out to protect his own planets.
I feel that Galactic empires is as good as,if not better than the star trek game which is available.
You will injoy playing this game and it will really go to your head when you start building your empire.Scott has done a good job writing this program,as he has done with many others.
TOP MARKS FOR THIS ONE SCOTT

Available from :-

Scott Le Brun
59 Brentwood Dve
Wantirna
VIC 3152 Australia

SEE BACK PAGE FOR SCOTTS
LATEST SOFTWARE PRICE LIST

INTRODUCTION TO PROGRAMMING

PART 1 - BY BOB KITCH

I have been asked to contribute a series on BASIC programming for the VZ Users. So here goes.

Firstly, the series will be unconventional. Most introductions to BASIC proceed blow-by-blow through the various BASIC commands. I will not - many texts exist which can explain these better than I can.

Secondly, the series will initially be non-specific to any particular computer language. General programming concepts and guidelines will be offered. The principles will be equally applicable to BASIC, Assembler, Pascal or whatever.

Thirdly, advanced programming concepts and hints will be offered as they are needed. This is the best time to introduce these since their mystique is removed.

Fourthly, early emphasis will be on PLANNING, ORGANISING and MAINTAINING a program, rather than encouraging feverish coding at the keyboard (which is usually commenced too early by beginners).

It is quite possible to recognise a breed of compulsive programmers, born from the home micro boom. This breed, is emerging from the brave new world of tomorrow's technology whose reason for existence is simply to program. People become totally fascinated by the unlimited abstract world that the inside of a computer offers. We can create a Universe or any World inside a machine. In the abstract world of programming, a well thought out programming method serves as a MAP, and the techniques of software engineering are the WEAPONS. These then are the main threads of this series.

Let's commence this month with a few definitions and concepts to ponder over until the next installment.

THE COMPUTER is a machine, and is only capable of doing simple work. It has been termed by some as "a remarkably efficient counting machine with a large memory - but no brains!" It has no intelligence and cannot think.

A COMPUTER SYSTEM consists of four elements :-

1. the Central Processor Unit (in the VZ it is the Z-80A microprocessor chip) with "primary memory" (ROM and up to 34K RAM).
2. Input devices - keyboard, cassette, disk and so on.
3. Output devices - screen, printer, cassette, disk, in-built speaker, voice and sound synthesisers etc.
4. "Secondary memory" - not essential but may be cassette or disk when used to update or relieve primary memory.

MAN-MACHINE INTERFACE. The interaction between man-machine inputs and outputs is a continuous and circular feedback process. e.g. man output (keypress) is machine input ..or.. machine output (screen prompt) is man input- ..and so on. This interaction forms the basis of using computers.

The four fold subdivision of a computer system is little different to our own mental capabilities. The CPU and primary memory is broadly equivalent to our mind. The I/O devices are similar to our senses (touch, taste, sight, sense of heat, speaking, hearing). The secondary memory is directly comparable to our use of external aids to assist our memory, such as note books, filing cabinets of information, telephone directories - all of which have slow access and are difficult to recall compared with things already resident in our mind.

COMPUTER PROCESSES or CAPABILITIES are surprisingly few in number. There are only FOUR and unless an exercise or problem can be broken down into these elementary processes, then coding of the program should not commence. A greater understanding of the problem is required before proceeding.

4 INTRODUCTION TO PROGRAMMING

It is important to clearly distinguish two things whilst programming. The first, is to devise a LOGICAL solution to the programming exercise, which is quite independent of the particular language to be used. The second, is the actual CODING of the exercise being undertaken. The latter stage is easy, provided that the former is well understood. The computer program will only function correctly if the logic of the program is correct, and there are no aids or diagnostics available from the machine to assist in achieving correctness in this demanding aspect of program design. Some diagnostics are however available to assist in the coding portion of the task - such as the SYNTAX checking.

As one becomes more familiar with programming languages it is soon apparent that many of the powerful command structures are simply macro instructions formed from these few "primatives".

The four processes are :-

1. Input data and store it in primary memory - the data may be either "raw" data input (e.g. from keyboard) or read-in from the secondary store. (e.g.tape).
2. Output data already stored in primary memory - either as "output" (e.g. to screen) or written-out to secondary memory (e.g.tape).
3. Perform simple arithmetic procedures (addition or subtraction) upon data in primary memory only.
4. Perform logical comparisons (disjunction, conjunction and negation) between two items of data in primary memory.

(Remember - I/O, arithmetic, comparisons only)

To continue the analogy with ourselves, I doubt whether we can do anything more than these operations except that we use experience. The computers' analogue of this is the PROGRAM as it possesses zero intelligence.

THE PROGRAMMING TASK is to utilize the high speed and large memory capacity of a computer system to do something useful - such as carry out calculations (number crunching), play games, monitor house security etc.

The spectrum of tasks involved in programming is very broad, so little wonder that beginners have trouble grasping the essentials, or that many programs are "badly" written. The task involves taking an idea or concept and translating that into a symbolic (program statement) form of representation. An intermediate stage in this translation often involves modelling the phenomenon being programmed. This psychologically involves moving from concrete concepts to various levels of abstraction - again a very difficult thing for, particularly young, minds to master.

The transition from an idea to a program can seldom be achieved in one leap - more often a number of intermediate steps are required. Liken it to writing an essay where drafts and notes are used before the final prose is produced. Fortunately a number of useful tools have been developed to assist in producing a good program.

In my view, one of the greatest pitfalls of the home computer boom is that these intermediate steps are not understood by Users so that, at least, bad programs and, at worst, disillusioned programmers result. Many of these people may find their way into the computer industry of the future. There is ALWAYS more personal satisfaction in achieving a "good" job even if it is only a games program for the kids. It is also more fun, (the essence of home micros) as there is less hassle in getting a program to run, and more time for more programs.

In the microcomputer environment where there are always hardware limitations, it means that it is very difficult to completely seperate hardware and software aspects of the programming task. The programmer may have to get "close to the hardware" - usually due to hardware/memory limitations or restricted I/O capabilities. Don't shy away from hardware by saying "but I am only interested in writing programs" as the two are somewhat inseperable.

Next month we will look at the various stages in the programming task, or how to approach a programming exercise. (see, no mention of BASIC code in this article!)

Finally, I would like to offer to Users that your programming queries will be answered if you write to me - with a SAE. please. In this manner you should get what you want and I will obtain a feel for the type of problems Users are experiencing.

WRITE TO:-

BOB KITCH,
7 EURELLA ST.,
KENMORE. QLD. 4069.

AUSTRALIAN VZ USERS GROUPS

VZ USER---Mark Harwood P.O Box 154 Dural N.S.W 2158

LE'VZ OOP---John D'Alton 39 Agnes St. Toowong Qlnd 4066

VZ DOWN UNDER---Scott Le Brun 59 Brentwood Dve Wantirna 3152

HUNTER VALLEY VZ USERS Group---P.O BOX 161 Jesmond N.S.W 2299

WAVZ---Graeme Bywater P.O BOX 388 Morley W.A 6062

VZ ASSEMBLY

You may have noticed that during this series on Assembly Language programming, we have often compared a Z80 instruction to an equivalent BASIC instruction. Carrying on this idea, what we'll be looking at this edition is similar to IF...THEN in BASIC. But first, let's look at the thing they call the Stack.

We said at the start that in some ways the Z80 registers are like BASIC variables. It should be obvious now that there are many limitations to this, one of which being the limited number of registers available, when compared to what BASIC offers. To help cope with this restriction, the Z80 offers what is called the stack. The stack is an area of RAM set aside for temporary

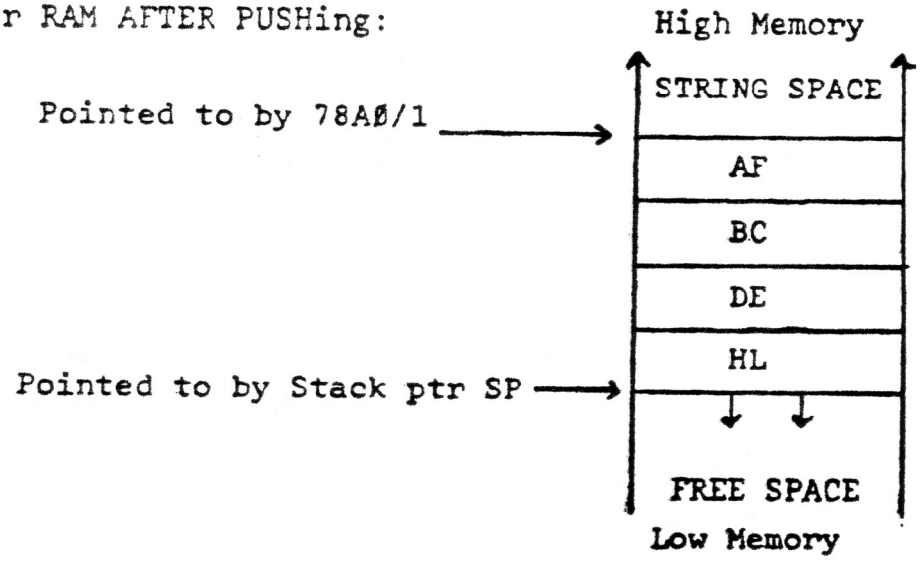
storage of the registers, and for storage of RETurn addresses after a CALL. Imagine that you are half way through a large assembly routine, and each register contains some vital number. However, you need to make a CALL to some smaller routine which uses these registers. If you were to simply CALL the routine all those important values would be lost. You need some way of 'saving' these values while you CALL the minor routine, where the registers will be used, then get back the important values into the registers. This is done using the stack operations, PUSH and POP.

To place the contents of one of the registers on the stack, PUSH is used. For example, PUSH HL puts the value stored in HL onto the stack. Think of the stack as a stack of plates, piled up vertically. Each plate represents an 8-bit number. When HL is PUSHed onto the stack, two more plates (HL is 16-bit) are placed on top of the pile. A special register, the stack pointer (SP) keeps track of where the top of this stack is in memory.

To retrieve values off the stack, the POP instruction is used. When a POP is executed, the top 'plate' of the stack is taken off and placed into a register. Obviously, this must be the value that was last PUSHed onto the stack. To make all this a little clearer, let's look at an example. Here's a short routine to illustrate.

```
START  PUSH AF          ;save registers on stack
       PUSH BC
       PUSH DE
       PUSH HL
       CALL  PROG       ;call routine using reg.
       POP HL           ;get registers back
       POP DE
       POP BC
       POP AF
```

User RAM AFTER PUSHing:



Firstly, all of the registers are pushed onto the stack, one at a time. Now stack space in the VZ is located below String Space, near the top of memory. As each register is PUSHed onto the stack, the stack pointer moves downward in memory. Thus, each subsequent register is stored below the previous one. Note that after HL has been PUSHed, the return address of the CALL PROG. command will also be stored on the stack, and then removed when the RET (return from routine) instruction is encountered.

Now the values are POPed off the stack, one at a time, in the reverse order that they were PUSHed. As each POP occurs, the

stack pointer moves upward in memory. In this way, the contents of registers can be preserved while another routine is executed which uses the registers.

The Stack can be of use to the assembly programmer in other ways. For example, suppose you wish to transfer the contents of one register into another, say copy the contents of AF into BC. This can be done with two simple stack operations:

```
PUSH AF     ;put AF on stack
POP BC      ;put into BC
```

While we're on the subject of swapping registers around, one useful instruction is EX DE,HL which swaps the contents of DE & HL registers.

Remember that we started out by commenting on the limited number of registers available. You may recall that a couple of months ago when registers were first introduced, we mentioned the existence of an alternate register set. This consists of an identical set of registers, completely independent of each other. (Only AF,BC,DE,HL are duplicated). To switch between each set, the EXX instruction is used. Programmers should be careful when using the prime and non-prime sets as it can become confusing as to which set is active at any particular time. Nevertheless, the alternate set can be handy when all the registers are being used by a CALLed routine. Just remember to switch back to the original set after RETing with another EXX.

You are probably wondering how to tell which registers will be used by a routine that you wish to CALL, when the routine is too long to look through. Usually the documentation indicates which registers are used by the given routine. An example of this can be found in the VZ Technical Manual, where listings of ROM routines are provided. These are very useful for the Assembly Language programmer. They allow the programmer to save memory, by not having to write some standard routines, such as clearing the screen, sounding the buzzer etc. When the programmer wants to carry out the function, he/she can simply CALL the routine which already exists in ROM as part of the BASIC operating system. Before doing so, however, registers which will be used by the ROM routine should be saved on the stack, and then POPed back after RETurning from the routine. Some of these routines require registers to contain some special value (called a parameter) before the routine is called. Many other useful routines are found in DOS (Disk Operating System) - consult the VZ300 Technical Manual.

THE STATUS FLAG

Bit	7	6	5	4	3	2	1	0
	SIGN FLAG	ZERO FLAG		HALF CARRY FLAG		PARITY/OVERFLOW FLAG	+/- FLAG	CARRY FLAG
Sign:	S	Z		H		P/V	N	C

The diagram above shows the layout of the F Register, or status flag of the Z80. Up until now, we have avoided mentioning it, but the time has come. You'll see that the status register is in some ways the most essential register of the CPU, as it allows testing of results of operations and conditional branching etc. Without it, the CPU would be unable to support any decision-making operations, which are essential to program structure in any

language. Basically, one or more of these flags are affected after some testing or arithmetic operation. We'll look at each of the flags, and the op-codes that are usually associated with them in turn.

One of the most commonly used flags is the zero flag. This flag can be thought of as an indicator for either a zero result from an addition or subtraction operation, or a match-up during a compare. To see how this comes about, we need to know a couple of new op-codes, CP and BIT.

CP stands for ComPare. In a compare, the specified data (this may be simply numeric or another register) is compared to the contents of the A register. For example: CP 20H will compare the contents of the A register with 20H. If the A reg. equals 20H, then the zero flag will be set to zero. If not, the zero flag is set to 1 (refered to as 'non-zero'). In this way the zero flag can be used to determine whether two values are equal. You may be interested to know that the Z80 actually subtracts the value from the A reg. to determine if they are equal. The result, however, is discarded and is therefore unimportant. Let's look at an example:

Here's the routine from last month:

```
001 LOOP   LD HL,50      ;set 1st frequency
002        LD BC,500     ;set duration
003        CALL 345CH    ;call sound routine
004        LD HL,60      ;set 2nd frequency
005        LD BC,500     ;set duration
006        CALL 345CH    ;call sound routine
007        JR LOOP       ; & repeat
```

Now here's an improved version:

```
001 LOOP   LD HL,50      ;set 1st frequency
002        LD BC,500     ;set duration
003        PUSH BC       ;save BC on stack
004        CALL 345CH    ;call sound routine
005        LD HL,60      ;set 2nd frequency
006        POP BC        ;get duration back from stack
007        CALL 345CH    ;call sound routine
008        CALL 2EF4H    ;scan keyboard
009        CP 0DH        ;was RETURN pressed?
010        JR NZ,LOOP    ;repeat sound if not
011        JP 31488      ;else return to EDASM
```

In the second routine we've made two important changes. Firstly, we made use of the stack to save the duration value while the sound routine was CALLed. This meant that there was no need to reset BC- we simply POPed it back off the stack.

Secondly, we added the option to stop the routine and get back to the command mode of the Ed.Asm. Notice the CALL 2EF4H. Just as CALL 345CH was a ROM routine to sound the speaker, so 2EF4H is a ROM routine which scans the keyboard for a keypress. If you consult your Technical Manual, you'll see that this routine loads the ASCII value of the key being pressed into the A reg. If no keys are being pressed, then A is loaded with zero. Although the routine disturbs all registers, we did not bother to save them on the stack beforehand, as there was no need to preserve them.

Now let's think again about what we want the program to do. The two-tone sound is produced until the RETURN key is pressed, when the routine must return to the Ed.Asm. Thus, after scanning the keyboard we need to test if the RETURN key is being pressed,

and if so, jump out of the routine and back to the Ed.Asm. This is similar to an IF...THEN...ELSE... statement in BASIC. First, the routine checks to see if the A reg. contains the ASCII code for RETURN. (ØDH). This is done using a CP instruction. Now as we mentioned previously, the zero flag is set according to whether a match is found. The next step is to act on the result of the compare. This is done using a variation on the JR (jump relative) op-code already introduced. In this case, JR NZ,LOOP is used, which literally means: 'if the zero flag is non-zero, then jump to the LOOP routine'. Notice that the letters NZ are used to indicate that the condition for jumping is 'non-zero'. Now, getting back to the routine, if the A reg. contained ØDH, then the zero flag would have been set to zero by the CP. Thus, the condition to jump back to LOOP would not have been met. Just like IF...THEN in BASIC, the Z80 goes on to the next instruction if a condition is not met. In this case, the next instruction is JP 31488, which sends control back to the Ed.Asm. Quite a few Z80 op-codes that we've already looked at support conditional operations. Some are listed below:

```
JR Z,       - Jump relative if zero
JR NZ,      - Jump relative if not zero
CALL Z,     - CALL routine if zero
CALL NZ,    - CALL routine if not zero
JP Z,       - Jump if zero
JP NZ,      - Jump if not zero
RET Z       - RETurn from routine if zero
RET NZ      - RETurn from routine if not zero
```

Note that these are only those concerned with the zero flag. Similar op-codes exist for many of the other flags we'll be looking at later.

Another instruction which is heavily dependent on the zero flag is BIT. BIT is used when one wants to test whether a particular bit in a particular register or address is 1 or Ø. For example, suppose we wanted to test directly whether the RETURN key was being pressed. If you consult the Tech. Manual, you'll see that when RETURN is pressed, bit 2 of location 68BFH is set to Ø. The Z80 does not allow us to test the address directly, so we must first load the address into HL then test bit 2:

```
LD HL,68BFH        ;get row address in HL
BIT 2,(HL)         ;test bit 2 of the address
                    pointed to by HL
JR NZ,LOOP         ;jump if not pressed
```

Bit 2 of the address pointed to by HL (68BFH) is tested, and the zero flag is set accordingly. If it is zero,i.e. if RETURN is pressed, then the routine continues on, otherwise it goes back to LOOP.

Numerous versions of the BIT instruction exist - refer to a Z80 op-code listing for more details. In most of the op-codes we introduce this will be the case, so keep one handy to familiarise yourself with what's supported.

Next month we'll be looking at one of the handiest op-codes around,DJNZ,and also at some of the other flags. Why not get hold of some listings of Z80 routines and see how much you know !

AEM'S CTS/SPO Speech Board - Program notes for the VZ 200/300

You will notice that the OUTPUT Routine for TALK A program is MUCH SHORTER and COMPACT when compared to my Original Programs.

In the routine 'TALK 3' the actual OUTPUT Routine takes a few lines, this is for Clarity Purposes.

In 'TALK A' you will see that the Output Routine has been reduced to just two lines.

Explanation of Output Routine taken from 'TALK 3'

```
10200 FOR T=1 TO LEN(A$) ' Starts Output Loop of A$

10220 A=ASC(MID$(A$,T,1)) ' A is made equal to the ASCII
      value of each letter in the String in turn

10240 OUT 12,A ' the ASCII value is output to the Printer.

10260 NEXT ' goes back for the next character in the String

10270 LPRINT:LPRINT:LPRINT ' forces the Speech to
      Output ALL of the String
```

OUTPUT Routine from 'TALK A'

```
1500 is the combination of lines 10200 to 10260 as above.
1520 is the same as 10270 above.
     I have found that only 2 LPRINT commands are needed.
```

Dave BOYCE

```
50 GOTO 100
60 ERA"TALK 3"
70 SAVE"TALK 3":DIR:STATUS
80 END
100 CLEAR 300 :' FILE - TALK 3
180 CLS:PRINT"TALKER VERSION 1.3"
200 PRINT
10110 ' SIGN ON
10120 A$="BY--YOR--COMMAND":GOTO 10180
10130 'INPUT SECTION
10140 PRINT" SEPERATE WORDS WITH A DASH"
10150 PRINT" E.G.->  HELLO-THERE-FRIEND":PRINT
10160 INPUT" TALK PT.1)";A$:INPUT" TALK PT.2)";B$:A$=A$+"-"+B$
10170 IF A$="-" THEN A$="PLEEZ-ENTER-SUM-THING-2-SAY"
10180 '
10190 ' OUTPUT LOOP ROUTINE
10200 FOR T=1 TO LEN(A$)
10220 A=ASC(MID$(A$,T,1))
10240 OUT 12,A
10260 NEXT
10270 LPRINT:LPRINT:PRINT
10280 GOTO 10140
10290 ' GO BACK TO SENDER
10300 END
```

```
10 CLS ' FILE - TALK A
15 ' DO NOT EDIT LINE 210
20 POKE31946,161
25 PRINT"   CTS/SPO SOUND EFFECTS DEMO.":PRINT :'FOR AEM 4505
30 CLEAR 300
35 PRINT"       1) GARGLING"
40 PRINT"       2) RUBBER LIPS"
45 PRINT"       3) TRAIN"
50 PRINT"       4) SNEEZES"
55 PRINT"       5) MACHINE GUN"
60 PRINT"       6) WHISPERING"
65 PRINT"       7) STARTER MOTOR"
70 PRINT"       8) SIREN (SORT OF)"
75 PRINT"       9) DOG"
80 PRINT"      10) OWN SOUND/MESSAGE"
85 PRINT"      11) UP-DATE DISK"
90 PRINT
200 PRINT@416,;:INPUT" ENTER YOUR CHOICE ";A
210 & A GOTO 500,550,600,650,700,750,800,850,900,950,1650
220 PRINT@437,"    ":GOTO 200
500 A$="GARGLING.ARRGLARRGLARRGLARRGLARRGLARRGLARRGL"
510 GOSUB 1500
520 GOTO 200
550 A$="RUBBER-LIPS.BLBLBLBLBLBLBLBLBLBL"
560 GOSUB 1500
570 GOTO 200
600 A$="STEAM-TRAIN.SHSHSHSHSHSS,TOOT-TOOT,SHSHSH"
620 GOSUB 1500
640 GOTO 200
650 A$="SNEEZING.ASJ,ASJ,ASJ,ASJ-GZUNTITE"
660 GOSUB 1500
670 GOTO 200
700 A$="MACHINE-GUN.GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG"
710 GOSUB 1500
720 GOTO 200
750 A$="WHISPERING.SHSH.SH..SH-SH-SH-SHSHSH-SH-SHSHSH"
760 GOSUB 1500
770 GOTO 200
800 A$="STARTER-MOTOR.IH-IH-IHIHIHIHIHIHIHIHIHIHIHIHIH"
810 GOSUB 1500
820 GOTO 200
850 A$="SIREN.OOWLOOWLOOWLOOWLOOWLOOWLOOWLOOOWLOOWL"
860 GOSUB 1500
870 GOTO 200
900 A$="DOGGY.WFFWFF-WFF-WFFWFF-RFF"
910 GOSUB 1500
920 A$="..THATS-PUHTHETIC"
930 GOSUB 1500
940 GOTO 200
950 INPUT" YOUR TRY ";A$
960 IF A$="",A$="YOU-DID-NOT-ENTER-ANY-THING.TRI--AGAN"
970 GOSUB 1500
980 '
990 RUN
1500 FOR T=1 TO LEN(A$):C=ASC(MID$(A$,T,1)):OUT12,C:NEXT
1520 LPRINT:LPRINT
1560 PRINT@437,"   ":A$="":RETURN
1600 END
```

```
1650 ' NO# 11 UPDATE DISK
1660 A$="DO-YOU-WAN-2-UP-DATE-THE-DISK":GOSUB 1500
1670 D$=INKEY$:D$=INKEY$
1680 IF INKEY$="N",220
1690 IF INKEY$="Y",1750
1700 IF INKEY$<>"Y" OR INKEY$<>"N",1670
1740 END
1750 ERA"TALK A":SAVE"TALK A":GOTO 220
2000 FOR L=31870 TO 32000
2020 PRINTL;PEEK(L);CHR$(PEEK(L))
2025 IF INKEY$=" ",2025
2030 NEXT
2040 LIST 20
```

INKEY$ INPUT ROUTINE BY PAUL LEON

As most programmers know the INPUT command on the VZ has it's limitations, but an INKEY$ INPUT routine can be tailored to your requirements. The routine below will not accept leading spaces, INVERSE or GRAPHIC characters. POKE30776,40 disables latter two inputs. Both ALPHA and NUMERICS are catered for. NC=VAL(IN$) in line 90 converts STRING to a NUMERIC value and ulike the INPUT command COMMA'S, etc are ignored. Imagine no more REDO's.

LE is the variable which sets the lenght of the INPUT. Only lines 200 to 310 are required, the rest are for demonstration purposes. Trailing spaces can be removed by line 300 or left in.

```
10 '*************************************************************
12 '*  INKEY$ INPUT ROUTINE DESIGNED BY PAUL LEON FOR HIS DAD  *
14 '*************************************************************
18 :
20 CLS:POKE30744,96:COLOR,0:POKE30862,80:POKE30863,52
25 SP$="                                    ":REM 30 SPACES
30 PRINT@39,"  INKEY$ INPUT  ":PRINT
35 PRINT"  DEMONSTRATION ROUTINE  ":PRINT:PRINT
40 PRINT"  USE RUBOUT (;)  KEY TO ERASE":PRINT
45 PRINT"  PREVIOUS ENTERED  CHARACTERS":GOTO65
50 PRINT@385," LIKE TO TRY AGAIN - Y/N  ";:SOUND30,1:LE=1
55 GOSUB200:Y$=IN$:IFY$="Y"THEN65ELSEIFY$="N"THEN60ELSE50
60 CLS:END
65 PRINT@417,SP$:GOSUB105
70 PRINT@385," REMOVE BLANK SPACES - Y/N  ";:SOUND30,1:LE=1
75 GOSUB200:YN$=IN$
80 IFYN$="Y"THEN85ELSEIFYN$="N"THEN85ELSEGOSUB110:GOTO70
85 GOSUB110:PRINT@385," NO. CHARS FOR TEXT - 16  ";:LE=2
90 GOSUB200:NC=VAL(IN$):IFNC<1ORNC>16THEN85ELSEPRINT@385,SP$
95 PRINT @385," ENTER TEXT  ";:SOUND30,1:LE=NC:GOSUB200
100 PRINT@417,SP$:PRINT@429,CHR$(34)IN$CHR$(34):GOTO50
105 PRINT@385,SP$:SOUND30,1:RETURN
110 PRINT@412,"    ":SOUND30,1:RETURN
180 :
190 REM" INKEY$ INPUT ROUTINE
200 CU$=CHR$(222)+CHR$(8):BS$=CHR$(8)+CHR$(127)+CU$:IN$=""
210 L=LEN(IN$):PRINT IN$;CU$;
220 A$=INKEY$:A$=INKEY$:IFA$=""THEN220ELSEX=USR(X)
230 POKE30776,40:IFA$=CHR$(13)THENPRINT" ":GOTO290
240 IFA$=";"ANDL>0THENPRINTBS$;:L=L-1:IN$=LEFT$(IN$,L)
250 IFL=LE THENSOUND20,1ELSEIFA$=" "ANDL=0THEN220
260 IFA$<" "ORA$>"^"ORA$=";"THEN220
270 IFL<LE THENPRINTA$;CU$;:IN$=IN$+A$:L=L+1
280 GOTO220
290 IFYN$="N"THENRETURN
300 IFRIGHT$(IN$,1)=" "THENIN$=LEFT$(IN$,LEN(IN$)-1):GOTO300
310 RETURN
```

SIDEWAYS BY DAVE BOYCE - - -

This Sub-Routine reads the high resolution (MODE(1)) Screen from the Bottom to top and from left to right.
AS it reads the screen,it prints it to the Dot Matrix Printer.The result is the screen is printed down the page instead of across the page.
E.G Like turning your screen on it's right side.

The D.M.P should respond to ESC Code 27 for this sub-routine to work.If your printer uses different ESC Codes then change lines 15510 and 15575 to the ESC Code used by your printer.

In use your main program would draw the HI-RES screen and then GOTO or GOSUB to line 15510 .After the screen dump is finished Line 15580 should send you back to the main program Via a GOTO or RETURN statement.

```
15470 ' SIDEWAYS PRINT-OUT SUB
15480 ' HI-RES COPY FOR M-1009
15490 ' OR SIMILAR EPSON COMPAT
15500 ' BY DAVE BOYCE  ADELAIDE
15510 LPRINTCHR$(27);"A";CHR$(6);
15520 FOR Y%= 0 TO 127
15530 FOR X%=63 TO 0 STEP-1
15540 P=POINT(Y%,X%)
15550 IF P=1 THEN LPRINT" ";:NEXT:GOTO 15570
15552 IF P=2 THEN LPRINT"Y";:NEXT
15554 IF P=3 THEN LPRINT"B";:NEXT
15560 IF P=4 THEN LPRINT"R";:NEXT
15570 LPRINT:NEXT
15575 LPRINTCHR$(27);"2"
15580 GOTO 000  'RETURN TO MAIN
15590 END
```

NZ TWISTER AGAIN - - -

Here are some super changes that can be made to Richard Davis's NZ TWISTER program which appeared in the last issue.
By making these changes you can convert the program to a sliding block puzzle instead of twisting.By using the same grid setup you slide a block into the blank squares to reassemble the map.Also by changing the DATA statements you can change the puzzle picture.

DELETE FROM LINE 14 TO 24 of NZ TWISTER
THEN REPLACE WITH....

```
50 Z$=INKEY$:Z$=INKEY$:IFZ$=""THEN50
51 IFZ$=" "THEN 9
52 IFZ$="S"THEN 12
54 SOUND15,4:Z=ASC(Z$)-64
56 W$=INKEY$:W$=INKEY$:IFW$=""THEN56
58 SOUND15,4
60 W=ASC(W$)-64:IFB(W)<>9THENSOUND17,9:GOTO56
62 C=B(W):B(W)=B(Z):B(Z)=C
64 GOSUB210:GOTO50
```

THEN CHANGE LINE 90 TO---

```
90 GOSUB210
```

VZ 200/300 ASSEMBLY LANGUAGE PROGRAMMING MANUAL FOR BEGINNERS.

This 140 page Manual has taken STEVE OLNEY over 18 months to write and is for sale at $24.95. I purchased my copy from :-

S.R.OLNEY P.O. Box 125 North Richmond N.S.W. 2754.

Below is a condensed version of the Contents to give some idea of the areas covered.

CHAPTER 1 - INTRODUCTION TO MACHINE CODE - Page 1.
CHAPTER 2 - PARTS OF YOUR VZ COMPUTER - Page 7.
CHAPTER 3 - MACHINE CODE and Hexadecimal Numbers - Page 11.
CHAPTER 4 - LOADING MACHINE CODE PEEK, POKE USR - Page 17.
CHAPTER 5 - THE Z80 REGISTERS - Page 21.
CHAPTER 6 - HARDWARE - CPU, RAM, ROM Video - Page 27.
CHAPTER 7 - BASIC AND MACHINE CODE - Peaceful Co-existence.
CHAPTER 8 - ARITHMETIC OPERATIONS - PAGE 53.
CHAPTER 9 - THE VIDEO SCREEN - Messages Simple Graphics.
CHAPTER 10 - JUMPS, BRANCHES - (And More on Stack Operations).
CHAPTER 11 - EDITOR ASSEMBLER - Page 85.
CHAPTER 12 - PROGRAMMING TECHNIQUES - Page 95.
CHAPTER 13 - THE Z-80 INSTRUCTION SET - Page 101.
CHAPTER 14 - INPUT AND OUTPUT - The Real World Outside.
CHAPTER 15 - CONCLUSION - Where To Go From Here - Page 121.
APPENDIX 1 - Memory Maps ..... Page 123.
APPENDIX 2 - Z80 Registers ..... Page 126.
APPENDIX 3 - Z80 Pinout ..... Page 127.
APPENDIX 4 - Keyboard Layout ..... Page 128.
APPENDIX 5 - System Pointers ..... Page 129.
APPENDIX 6 - Common Z80 Opcodes ..... Page 131.
APPENDIX 7 - Hexadecimal/Decimal Tables ..... Page 138.
APPENDIX 8 - Z80 Mnemonics Recognised By The VZ Editor Assembler ..... Page 139.

I found the Manual to be of a decent layout with a good index giving full details of each chapter, the Manual is well presented using a plastic ring binder.

Information is for the VZ not just Z80 type Computers and I'm sure anyone that USES this Manual in conjunction with the Dick Smith Electronics VZ 200/300 Technical Reference Manual will master Assembly Language Programming, Its quite obvious that one has to try out the routines and examples given in the Manual to grasp the subjects covered.

On my experience so far working through the Manual I was impressed with Steve's methods of explanation. Contents of each chapter are readily found in the contents list. I would recommend to anyone finding this Manual useful to purchase a copy of Programming The Z80 by Rodney Zaks Published By Sybex Computer Books, this book will allow you to cover the use of the complete instruction set to use and design useful routines.

(R.Woods.)

SOFTWARE FOR SALE

Prices shown include postage and packing anywhere in Australia. New Zealand customers, please add A$2.00 to the prices below. Make sure that the memory of your VZ system is greater than or equal to the memory required by the software that you wish to purchase. Memory sizes indicated do not include Video Ram. Prices shown are for both tape and disk copies All prices are in Australian dollars.

* PLEASE NOTE *

SNAKE DUEL & STELLAR WARS have both been dropped from the range due to lack of sales.

VZ MONOPOLY V2.0 Both the 22K and 32K versions are included together, so that if you have a 22K system and have the 22K VZ MONOPOLY V2.0 and you wish to upgrade to the 32K version, you don't have to re-purchase the same program.

NEW SOFTWARE

FBI 2001

Welcome to the year 2001.
You as a future agent of the FBI have been called forth to capture a criminal within a certain time limit. Using your teleport, you track the crook around the world picking up clues to his identity and his next destination as you go. With this information you must get a warrent from HQ before you can legally arrest him or her.

After finishing a case you go up in the ranks. The ultimate rank is FBI-ELITE!

Overall an excelent program, that uses lo-res graphics to draw scenes of each location. Written by P.Frantz.

BLACKJACK ROYALE	- 16K	- $12.50
CASTLE GREYSTONE	- 32K	- $12.50
DATA	- 16K	- $20.00
FBI-2001	- 32K	- $15.00
GALACTIC EMPIRES	- 16K	- $12.50
GRAPHSTAR	- 22K	- $20.00
HAUNTED MANSION	- 22K	- $12.50
KNIGHTS QUEST	- 32K	- $15.00
NAMESTAR/SHOPSTAR	- 22K	- $12.50
SCOTLAND YARD	- 16K	- $6.50
TRIVIAL CULT	- 16K	- $6.50
VZ MONOPOLY V2.0	- 22K	- $15.00

ADVENTURER'S COLLECTION - Contains SCOTLAND YARD, HAUNTED MANSION, KNIGHTS QUEST & CASTLE GREYSTONE.

All on the one tape or 2 disks - $36.50

Please make all cheques and money orders payable to :-

Scott Le Brun
59 Brentwood Dve
Wantirna VIC 3152